

DATA CLEANING & DATA PREPARATION PLAYBOOK

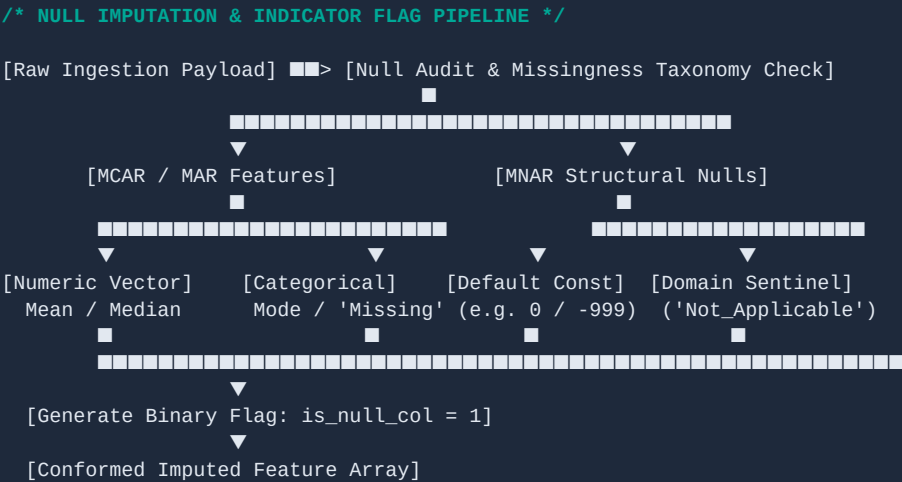
PRODUCTION DESIGN PATTERNS, QUALITY GUARDRAILS & ARCHITECTURAL DIAGRAMMS

Key Execution Rules: Executive Summary: Data cleaning and data preparation represent the cornerstone of production data engineering. This playbook outlines 8 mission-critical architectural design patterns for handling dirty, missing, duplicated, and anomalous data across enterprise Lakehouses, real-time feature pipelines, and high-performance analytical query engines.

TOPIC 01 // DATA QUALITY & MISSINGNESS

Null Hygiene & Imputation Architecture

Missing data in production systems is governed by three primary mechanisms: **Missing Completely at Random (MCAR)**, **Missing at Random (MAR)**, and **Missing Not at Random (MNAR)**. Simply dropping rows or blindly filling mean values introduces severe selection bias and destroys feature distributions. The standard enterprise pattern pairs statistical/k-NN imputation with explicit binary indicator flag columns (e.g., `is_null_feature`) to preserve the missingness signal for downstream ML models.

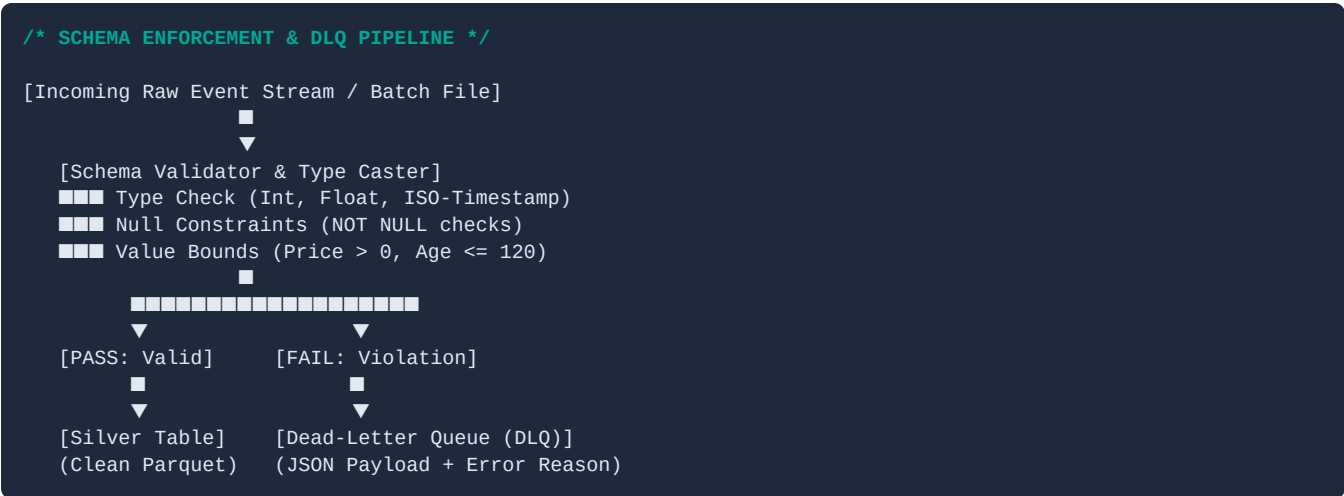


Key Execution Rules: 1. Never drop rows without auditing missingness percentage (<5% threshold for listwise deletion). 2. Always append a binary flag column (`col_isnull`) before imputing numeric features to maintain model variance awareness. 3. Compute imputation parameters (mean/median/k-NN weights) strictly on training split to eliminate data leakage.

TOPIC 02 // DATA CONTRACTS & INGESTION

Schema Enforcement & Dead-Letter Quarantine

Data contracts and declarative schema validation (using frameworks like Pandera, Great Expectations, or Delta Schema Enforcement) prevent corrupted upstream payloads from polluting production tables. Bad records are routed to a **Dead-Letter Queue (DLQ) / Quarantine Bucket** for asynchronous triage while valid records proceed seamlessly down the pipeline.



Key Execution Rules: 1. Enforce strict schema validation at the landing layer (Bronze to Silver transition). 2. Store quarantine records in raw original format alongside error metadata (timestamp, failed constraint, raw payload). 3. Trigger alerts if Quarantine DLQ volume exceeds 1% of total batch ingestion.

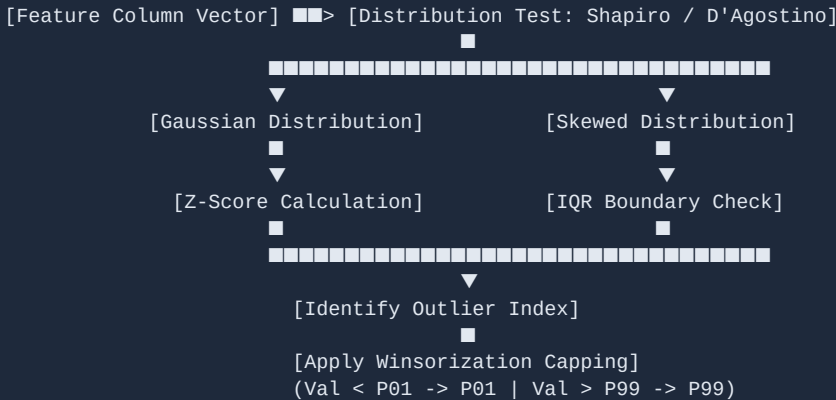
TOPIC 03 // STATISTICAL CLEANING

Outlier Detection & Robust Winsorization

Outliers distort statistical aggregations and cause gradient explosion in linear models and neural networks. Depending on distribution symmetry, engineers choose between **Z-Score** ($|Z| > 3.0$), **IQR Bounds** ($[Q1 - 1.5 \cdot IQR, Q3 + 1.5 \cdot IQR]$), and multi-dimensional **Isolation Forests**. Rather than dropping extreme outliers, **Winsorization** caps values at preset percentile thresholds (1st & 99th percentiles).

Method	Applicable Distribution	Formula / Logic	Recommended Action
Z-Score Method	Gaussian / Unimodal	$ Z = (X - \mu) / \sigma > 3.0$	Cap at $\mu \pm 3\sigma$
IQR Bounds	Skewed / Non-Parametric	$Q1 - 1.5 \cdot IQR$ to $Q3 + 1.5 \cdot IQR$	Winsorize to 1st/99th Pct
Isolation Forest	Multivariate Anomalies	Random partitioning depth	Flag as Anomaly Row

/* STATISTICAL OUTLIER DETECTION & WINSORIZATION */



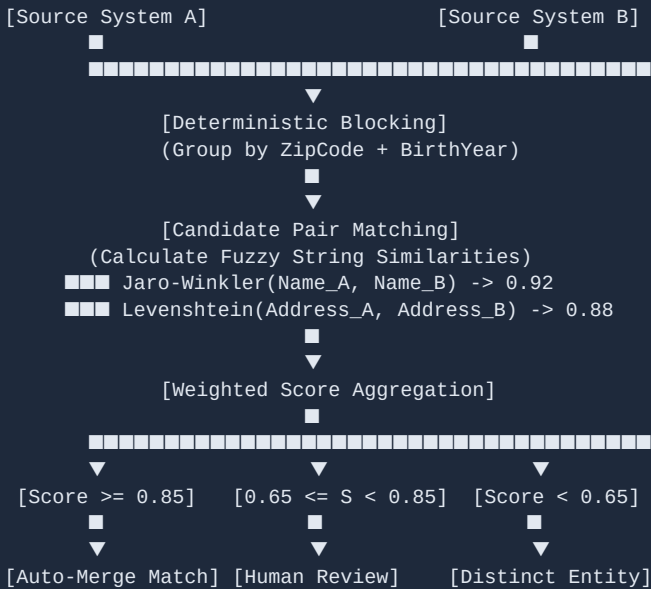
Key Execution Rules: 1. Never use Z-score on heavily skewed or multi-modal distributions; default to IQR or Isolation Forests. 2. Prefer WinsORIZATION over deletion to maintain dataset row count and sample alignment. 3. Store raw original values in audit attributes if downstream regulatory compliance requires non-destructive edits.

TOPIC 04 // ENTITY RESOLUTION

Probabilistic Record Linkage & Deduplication

Duplicate customer or entity records across disparate source systems rarely share exact primary keys. The **Fellegi-Sunter Probabilistic Record Linkage** framework combines deterministic blocking keys (e.g., Zip Code + Gender) with fuzzy string distance metrics (**Jaro-Winkler**, **Levenshtein Distance**) to match entities without $O(N^2)$ exhaustive comparisons.

/* PROBABILISTIC DEDUPLICATION & RECORD LINKAGE */



Key Execution Rules: 1. Always perform blocking to reduce comparison space from quadratic $O(N^2)$ to linear $O(N \cdot B)$. 2. Use Jaro-Winkler for short strings (person names) and Levenshtein / Token Sort Ratio for addresses. 3. Establish a three-tier decision boundary: Auto-Match, Human-in-the-loop Review Queue, and Non-Match.

TOPIC 05 // TEXT CLEANING

String Normalisation & Categorical Standardization

Unstructured and user-entered text fields harbor typos, irregular casing, white space noise, and non-standard symbols. Text standardization pipelines execute ASCII lowercasing, unicode normalization (NFC/NFKD), regex whitespace stripping, and canonical dictionary mapping before categorical encoding.

Raw Input String	Cleaning Step Applied	Cleaned Conformed Value
N.Y.C. , USA!	Regex strip punctuation & whitespace	nyc usa
François-123	Unicode NFKD strip accents & digits	francois
+1 (555) 019-2831	E.164 Regex Phone Standardisation	+15550192831
2026/10/16 10:21:29	ISO 8601 Timestamp Parsing	2026-10-16T10:21:29Z

```
/* STRING NORMALISATION & CANONICAL MAPPING */

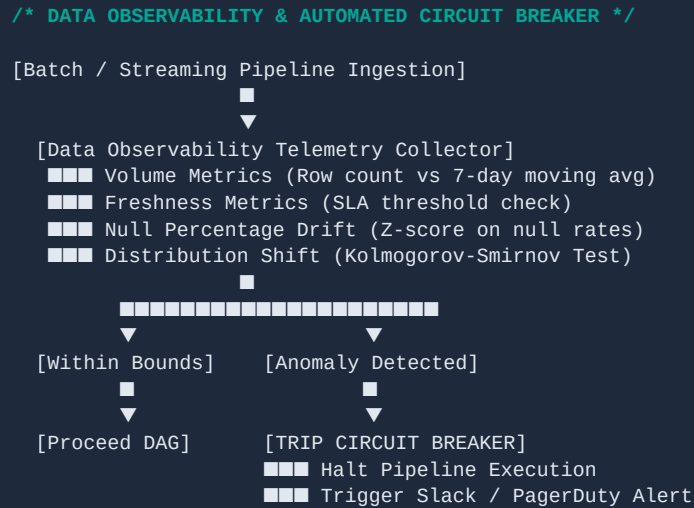
[Raw Unstructured Text Field]
  ▼
[Unicode NFKD Normalization] > Strip Diacritics / Accents
  ▼
[Regex Lowercasing & Punctuation Strip]
  ▼
[Canonical Synonym Dictionary Map] (e.g. 'NY' -> 'New York')
  ▼
[Frequency-Based High-Cardinality Grouping] ('Other' Binning)
  ▼
[Clean Conformed Categorical Feature]
```

Key Execution Rules: 1. Apply Unicode NFKD normalization to eliminate accents and special symbols safely. 2. Group low-frequency categorical levels (<1% dataset share) into an 'Other' bucket to prevent curse of dimensionality. 3. Use ISO 8601 formats for all datetime strings across analytical data stores.

TOPIC 06 // PIPELINE GUARDRAILS

Data Observability & Anomaly Monitoring

Data Quality is not a static one-time script; it requires continuous monitoring. Data Observability engines track 5 key pillars: **Volume**, **Freshness**, **Schema Shift**, **Null-Rate Drift**, and **Distribution Anomaly**. When anomalies occur, automated circuit breakers halt downstream DAG execution.

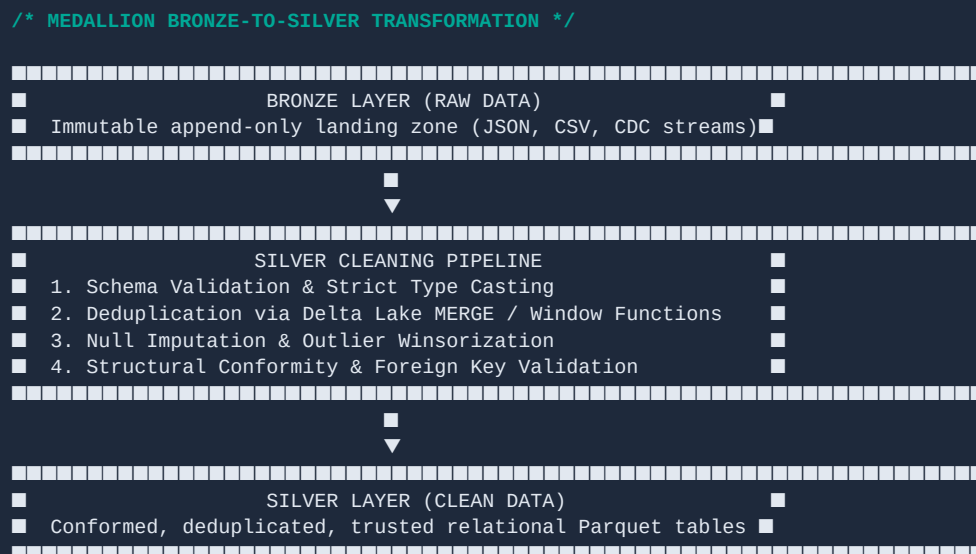


Key Execution Rules: 1. Set volume alerts on 7-day rolling moving averages with 3-sigma bounds. 2. Implement Kolmogorov-Smirnov (KS) tests on continuous numerical features to catch distribution shift before model scoring. 3. Fail-safe: Always stop pipeline propagation if null percentage spikes by >15% over baseline.

TOPIC 07 // LAKEHOUSE ARCHITECTURE

Medallion Cleaning Pipeline (Bronze to Silver)

In the Medallion Lakehouse Architecture, the **Bronze layer** maintains an immutable append-only record of raw data. The **Silver layer** executes the full data cleaning lifecycle: schema enforcement, type casting, deduplication, referential integrity validation, and business logic normalization.



Key Execution Rules: 1. Never execute business aggregation or metrics calculations in the Silver layer (reserve for Gold). 2. Ensure Silver transformations are idempotent: re-running a batch must produce the exact same Silver state. 3. Maintain raw source metadata fields (`_ingested_at`, `_source_file`) for lineage auditing.

TOPIC 08 // HIGH-PERFORMANCE COMPUTING

High-Performance Cleaning: Polars vs Pandas

Traditional Pandas relies on single-threaded Python execution and expensive memory copies over NumPy arrays. **Polars** leverages a Rust multi-threaded query engine, Apache Arrow in-memory columnar layout, and **Lazy Evaluation** to optimize filter and cleaning queries before execution.

Feature / Dimension	Pandas (Eager)	Polars (Lazy Engine)
Memory Format	Custom NumPy / Python Objects	Apache Arrow Columnar Format
Execution Engine	Single-threaded Eager	Rust Multi-threaded Work-Stealing
Query Optimization	None (Executes step-by-step)	Predicate & Projection Pushdown
Memory Footprint	High (3-5x dataset size copy)	Zero-Copy Arrow Slices

```
/* POLARS LAZY QUERY OPTIMIZATION ENGINE */

[Parquet Storage Files]
  ▼
[polars.scan_parquet()] <■■■ Lazy Frame Plan Construction
  ▼
[Polars Query Optimizer]
  ■■■ Predicate Pushdown (Filter at file scan level)
  ■■■ Projection Pushdown (Read only required columns)
  ■■■ Expression Folding (Combine string cleaning steps)
  ▼
[Multi-threaded Rust Execution Engine] (Zero-Copy Arrow Memory)
  ▼
[Cleaned Polars DataFrame]
```

Key Execution Rules: 1. Use `pl.scan_parquet()` lazy evaluation over `pl.read_parquet()` eager loading for large datasets. 2. Chain cleaning expressions inside a single `.with_columns()` block to enable Polars parallel execution. 3. Avoid falling back to Python `.map_elements()` lambdas inside Polars; use native expression syntax.